目錄

Java 面试笔记

这本书其实是我的一本笔记 (还在整理中). 我是也是刚找到工作. 这本笔记主要记录了我之前面试遇到的问题以及我在网上整理的一些资料 主要是面向 junior 级别的 就是我们这些小菜鸟啦 ~

目前只有我一个人 希望大家都能加入进来一起贡献或者是指出错误

如果有确实帮助到大家 可以进我的源代码在右上角 star 一下下 :)

Gitbook地址

Github托管地址

如何贡献

任何问题都欢迎直接联系我 dongchuanyz@163.com QQ 526402328

因为现在 gitbook 支持在线编辑功能. 所以想要贡献的同学可以直接联系我开权限！

许可证

Java

第一部分是 Java 的基础面试题 补充ing

说一说 Java

Sun 公司在 1995 创建

Java 的一些特点?

- Object Oriented 面向对象
- Platform Independent 平台独立
- Interpreted 解释性语言
- Multi-threaded 多线程

但是 Java 最重要的特点是平台独立

平台独立意味着我们可以在一个系统编译它然后在另外一个系统使用它

PS : 有一些中文的博客会说 java 也是编译性语言 因为国内博客都是你抄我 我抄你 所以你懂的 当然最好就是给出编译混合解释性这个说法 然后给出自己的理解 我自己在国外面试的时候也问了几个面试官关于这个问题 都说解释性语言是最准确的.

Java 为什么是高效的 (High Performance)?

因为 Java 使用 Just-In-Time (即时) 编译器.

把java字节码直接转换成可以直接发送给处理器的指令的程序.

列举出2个 IDE

Netbeans, Eclipse, etc.

面向对象的特征有哪些方面

- 封装

让变量和访问这个变量的方法放在一起，将一个类中的成员变量全部定义成私有的，只有这个类自己的方法才可以访问到这些成员变量

- 抽象

声明方法的存在而不去实现它的类被叫做抽象类

- 继承

继承是子类自动共享父类数据和方法的机制，这是类之间的一种关系，提高了软件的可重用性和可扩展性

- 多态

多态就是指一个变量, 一个方法或者一个对象可以有不同的形式.

JDK JRE JVM?

- 解释它们的区别
- 为什么 JVM 不是平台独立的

JDK

Java Development Kit 用作开发, 包含了JRE, 编译器和其他的工具(比如: JavaDoc，Java调试器), 可以让开发者开发、编译、执行Java应用程序.

JRE

Java 运行时环境是将要执行 Java 程序的 Java 虚拟机, 可以想象成它是一个容器, JVM 是它的内容.

JRE = JVM + Java Packages Classes(like util, math, lang, awt,swing etc)+runtime libraries.

JVM

Java virtual machine (Java 虚拟机) 是一个可以执行 Java 编译产生的 Java class 文件 (bytecode) 的虚拟机进程, 是一个纯的运行环境.

JVM 不是平台独立的

Java被设计成允许应用程序可以运行在任意的平台, 而不需要程序员为每一个平台单独重写或者是重新编译. Java虚拟机让这个变为可能, 因为它知道底层硬件平台的指令长度和其他特性.

Reference

Diff between JRE and JVM

什么是对象 (Object)?

- 对象是程序运行时的实体
- 它的状态存储在 fields (也就是变量)
- 行为是通过方法 (method) 实现的
- 方法上操作对象的内部的状态
- 方法是对象对对象的通信的主要手段

一个类是由哪些变量构成的?

- Local variable 本地变量
- instance variables 实例变量
- class variables 类变量

Local variable

在方法体, 构造体内部定义的变量 在方法结束的时候就被摧毁

instance variables

在类里但是不在方法里 在类被载入的时候被实例化

class variables

在类里但在方法外, 加了 static 关键字. 也可以叫做静态变量

静态变量和实例变量的区别？

- 在语法定义上的区别：静态变量前要加static关键字，而实例变量前则不加。
- 在程序运行时的区别：实例变量属于某个对象的属性， 必须创建了实例对象(比如 new 一个)， 其中的实例变量才会被分配空间， 才能使用这个实例变量. 静态变量不属于某个实例对象， 而是属于类， 所以也称为类变量， 只要程序加载了类的字节码， 不用创建任何实例对象, 静态变量就会被分配空间, 静态变量就可以被使用了.
- 总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用.

例如, 对于下面的程序, 无论创建多少个实例对象, 永远都只分配了一个staticVar变量, 并且每创建一个实例对象, 这个staticVar就会加; 但是, 每创建一个实例对象, 就会分配一个instanceVar, 即可能分配多个instanceVar, 并且每个instanceVar的值都只自加了1次.

```
public class VariantTest{

        public static int staticVar = 0;
        public int instanceVar = 0;

        public VariantTest(){
               staticVar++;
               instanceVar++;
               System.out.println(“staticVar=” + staticVar + ”,instanceVar=”+ instance
Var);
        }
}
```

封装 Encapsulation

- 使一个类的变量 private
- 提供 public 方法来调用这些变量. 所以外部类是进不去的. 这些变量被隐藏在类里了. 只能通过已经定义的 pulic 方法调用.

好处

当我们修改我们的实现的代码时, 不会破坏其他调用我们这部分代码的代码. 可维护性， 灵活性和可扩展

多态 Polymorphism

多态就是指一个变量, 一个方法或者一个对象可以有不同的形式.

多态主要分为

- 重载overloading

```
就是一个类里有两个或更多的函数，名字相同而他们的参数不同.
```

- 覆写overriding

```
是发生在子类中！也就是说必须有继承的情况下才有覆盖发生. 当你继承父类的方法时, 如果你感到哪个方法不爽，
功能要变，那就把那个函数在子类中重新实现一遍.
```

例子

重载 静态捆绑 (static binding) 是 Compile 时的多态

```
EmployeeFactory.create(String firstName, String lastName){...}
EmployeeFactory.create(Integer id, String lastName){...}
```

覆写 动态捆绑 (dynamic binding) 是 runtime 多态

```
public class Animal {
    public void makeNoise()
    {
        System.out.println("Some sound");
    }
}

class Dog extends Animal{
    public void makeNoise()
    {
        System.out.println("Bark");
    }
}

class Cat extends Animal{
    public void makeNoise()
    {
        System.out.println("Meawoo");
    }
}


public class Demo
{
    public static void main(String[] args) {
        Animal a1 = new Cat();
        a1.makeNoise(); //Prints Meowoo

        Animal a2 = new Dog();
        a2.makeNoise(); //Prints Bark
    }
}
```

构造器是否可以被 override

构造器不能被继承 所以不能被 override 但可以被重载 overload

接口 Interface

接口是抽象方法的集合。一个类实现一个或多个接口，因此继承了接口的抽象方法.

接口的特点

- 不能实例化
- 没有构造体
- 所有方法都是抽象的 (abstract).同时也是隐式的 public. 也就是说声明时, 可以省略 public abstract.
- 只能含有声明为 final static 的 field

接口和抽象的区别

抽象类可以有构造方法 接口不行

抽象类可以有普通成员变量 接口没有

抽象类可以有非抽象的方法 接口必须全部抽象

抽象类的访问类型都可以 接口只能是 public abstract

一个类可以实现多个接口 但只能继承一个抽象类

基础概念题

下面哪一项说法是正确的

```
1. 在一个子类里,一个方法不是 public 就不能重载
2. 覆盖一个方法只需要满足相同的方法名和参数类型
3. 覆盖一个方法必须方法名,参数和返回类型都相同
4. 一个覆盖的方法必须有相同的方法名,参数名和参数类型
```

答案 3

覆盖函数与被覆盖函数只有函数体不同

下面哪一项说法是错误的

```
1. 重载函数的函数名必须相同
2. 重载函数必须在参数个数或类型上有所不同
3. 重载函数的返回值必须相同
4. 重载函数的函数体可以不同
```

答案 3

函数的重载与函数的返回值无关

下面哪一项说法是正确的

```
1. 静态方法不能被覆盖
2. 静态方法不能被声明称私有
3. 私有方法不能被重载
4. 一个重载的方法在基类中不通过检查不能抛出异常
```

答案 1

基础程序题

题目一

```
class Base{}

class Agg extends Base{
    public String getFields(){
        String name = "Agg";
        return name;
    }
}

public class Avf{
    pulic static void main(String argv[]){
        Base a = new Agg();
        //here
    }
}
```

下面哪个选项的代码替换到//here会调用getFields方法,使出书结果是Agg

```
A. System.out.println(a.getFields());
B. System.out.println(a.name);
C. System.out.println((Base)a.getFields());
D. System.out.println(((Agg)a).getFields());
```

答案 D

Base 类要引用 Agg 类的实例需要把 Base 类显示地转换成 Agg 类,然后调用 Agg 类中的方法. 如果 a 是 Base 类的一个实例,是不存在这个方法的,必须把 a 转换成 Agg 的一个实例

题目二

```
class A{

    public A(){
        System.out.println("A");
    }
}

public class B extends A{

    public B(){
        System.out.println("B");
    }

    public static void main(String[] args){
        A a = new B();
        a = new A();
    }
}
```

输出结果是 A B A

题目三

```
class A{
    public void print(){
        System.out.println("A");
    }
}

class B extends A{
    public void print(){
        System.out.println("B");
    }
}

public class Test{
    ..
    B objectB = new B();
    objectB.print();

    A as = (A) objectB;
    as.print();

    A asg = objectB;
    asg.print();

    as = new A();
    as.print();
    ..
}
```

输出为 B B B A

题目四

```
public class Test {
    public static void main(String[] args){
        Father father = new Father();
        Father child = new Child();
        System.out.println(father.getName());
        System.out.println(child.getName());
    }
}

class Father{
    public static String getName(){
        return "Father";
    }
}

class Child extends Father{
    public static String getName(){
        return "Child";
    }
}
```

输出是 Father Father 因为这里的方法 getName 是静态的. 具体执行哪一个,则要看是由哪个类来调用的.

super 关键词

- 调用父类 (Superclass) 的成员或者方法
- 调用父类的构造函数

1. 调用父类 (Superclass) 的成员或者方法

如果你的方法覆写一个父类成员的方法, 你可以通过 super 关键字调用父类的方法. 考虑下面的父类:

```
public class Superclass {

    public void printMethod() {
            System.out.println("Printed in Superclass.");
        }
}
```

下面是一个子类 (subclass), 叫做 Subclass, 覆写了 printMethod():

```
public class Subclass extends Superclass {

    // overrides printMethod in Superclass
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
        Subclass s = new Subclass();
        s.printMethod();
    }
}
```

输出

```
Printed in Superclass.
Printed in Subclass
```

1. 调用父类的构造函数

使用 **super** 关键字调用父类的构造函数. 下面的 MountainBike 类是 Bicycle 类的子类. 它调用了父类的构造方法并加入了自己的初始化代码:

```
public MountainBike(int startHeight,
                    int startCadence,
                    int startSpeed,
                    int startGear) {
    super(startCadence, startSpeed, startGear);
    seatHeight = startHeight;
}
```

调用父类的构造体必须放在第一行.

使用

```
super();
```

或者:

```
super(parameter list);
```

通过 super(), 父类的无参构造体会被调用. 通过 super(parameter list), 父类对应参数的构造体会被调用.

注意: 构造体如果没有显式的调用父类的构造体, Java 编译器自动调用父类的无参构造. 如果父类没有无参构造, 就会报错 (compile-time error).

Super 程序题

题目一

```
class Base{
    Base(){
        System.out.println("Base");
    }
}

public class Checket extends Base{
    Checket(){
        System.out.println("Checket");
        super();
    }
    public static void main(String argv[]){
        Checket a = new Checket();
    }
}
```

输出是什么？ 是 compile time error. super() 必须放在前面.

放在前面之后,输出为 Base Checket

题目二

```
import java.util.Date;

public class Test extends Date{

    public static void main(String[] args) {
       new Test().test();
    }

    public void test(){
       System.out.println(super.getClass().getName());
    }
}
```

返回的结果是 Test

因为super.getClass().getName() 调用了父类的 getClass() 方法, 返回当前类

如果想得到父类的名称，应该用如下代码：

```
getClass().getSuperClass().getName()
```

题目三

```
public abstract class Car {

    String name = "Car";

    public String getName(){
        return name;
    }

    public abstract void demarre();
}

public class B extends Car{
    String name = "B";

    public String getName(){
        return name;
    }

    public void demarre() {
        System.out.println(getName() + " demarre");
    }
}

public class C extends B{
    String name = "C";

    public String getName(){
        return name;
    }

    public void demarreWithSuper() {
        System.out.println(super.getName() + " demarre");
    }

    public void demarreNoSuper() {
        System.out.println(getName() + " demarre");
    }
}

public class D extends B{
    public String getName(){
        return name;
    }

    public void demarreNoSuper() {
        System.out.println(getName() + " demarre");
    }
```

```
}

public class Test {
    public static void main(String[] args) {
        B b = new B();
        b.demarre();

        Car bCar = new B();
        bCar.demarre();

        C c = new C();
        c.demarre(); // c 里并没有定义这个函数
        c.demarreWithSuper();
        c.demarreNoSuper();

        D d = new D();
        d.demarre();

        transfer(c);    // TransferC
        transfer((B)c); // TransferB
        transfer(d);    // TransferB
    }

        public static void transfer(B b){
            System.out.println("TransferB");
            b.demarre();
        }

        public static void transfer(C c){
            System.out.println("TransferC");
            c.demarre();
        }
    }
}
```

输出是 B demarre B demarre C demarre B demarre C demarre B demarre TransferC C demarre TransferB C demarre TransferB B demarre

this 程序题

题目一

```
class Tester{
    int var;
    Tester(double var){this.var = (int)var};
    Tester(int var){this("hello");
    Tester(String s){
        this();
        System.out.println(s);
    }

    Tester(){ System.out.println("good-bye");}
}
```

Tester t = new Tester(5) 的输出是什么?

```
good-bye
hello
```

题目二

貌似和 this 无关但是很重要 public class Base { int i;

```
    Base(){
        add(1);
        System.out.println(i);
    }

    void add(int v){
        i+=v;
        System.out.println(i);
    }
}

public class MyBase extends Base{
    MyBase(){
        System.out.println("MyBase");
        add(2);
    }

    void add(int v){
        System.out.println("MyBase Add");
        i+=v*2;
        System.out.println(i);
    }
}

public class Test {
    public static void main(String[] args) {
        go(new MyBase());
    }

    static void go(Base b){
        b.add(8);
    }
}
```

输出的结果是 22

子类会首先调用父类的构造函数,在父类的构造函数 Base() 中执行 add() 方法. 但这个 add() 方法由于是在新建 MyBase 对象时调用的. 所以是执行的 MyBase 中的 add 方法

在Java中，子类的构造过程中，必须 调用其父类的构造函数, 是因为有继承关系存在时, 子类要把父类的内容继承下来, 通过什么手段做到的？ 这样： 当你new一个子类对象的时候, 必须首先要new一个父类的对像出来, 这个父类对象位于子类对象的内部, 所以说，子类对象比父类对象大, 子类对象里面包含了一个父类的对象, 这是内存中真实的情况.

构造方法是new一个对象的时候, 必须要调的方法, 这是规定, 要new父类对象出来, 那么肯定要调用其构造方法, 所以 第一个规则：子类的构造过程中，必须 调用其父类的构造方法

一个类, 如果我们不写构造方法, 那么编译器会帮我们加上一个默认的构造方法, 所谓默认的构造方法, 就是没有参数的构造方法, 但是如果你自己写了构造方法, 那么编译器就不会给你添加了

所以有时候当你new一个子类对象的时候，肯定调用了子类的构造方法，但是在子类构造方法中我们并没有显示的调用基类的构造方法，就是没写，如：super(); 并没有这样写，但是

第二个规则：如果子类的构造方法中没有显示的调用基类构造方法，则系统默认调用基类无参数的构造方法

注意：如果子类的构造方法中既没有显示的调用基类构造方法，而基类中又没有默认无参的构造方法，则编译出错，所以，通常我们需要显示的：super(参数列表)，来调用父类有参数的构造函数

抽象 abstract

Abstract 类

- 不能实例化

Abstract 方法

- 在父类里定义抽象方法,在子类里定义这个具体的方法,所以它是抽象的.

好处

减少复杂度和提高可维护性

abstract 相关问题

题目一

什么是抽象类

1. A class with no methods
2. A class with no concrete subclasses
3. A class with at least one undefiend message
4. None of above

答案是 4 我们是可以定义一个抽象空类的

```
abstract class emptyAb {}
```

题目二

```
abstract class Base{
    abstract public void myfunc();
}

public class Abs extends Base{
    public static void main(String argv[]){
        Abs a = new Abs();
        a.amethod();
    }

    public void amethod(){
        System.out.println("A method");
    }
}
```

运行的结果

1. 输出 A method
2. The code will compile but complain at run time
3. The compiler will complain errors in Abs class

答案是 3

this() 和 super() 在构造体里怎么用?

this() 在同一个类的构造体被调用. this("toto","tata",1)相当于调用对应参数的构造体.

super() 用来调用父类构造体.

Static 关键字

Static 关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例的情况下直接被访问

声明为 **static** 的方法有以下几条限制：

1. 仅能调用其他的 static 方法
2. 只能访问 static 变量.
3. 不能以任何方式引用 this 或 super
4. 不能被覆盖.

声明为 **static** 的变量实质上就是全局变量. (+ final 就是全局常量). 当声明一个对象时, 并不产生 static 变量的拷贝, 而是该类所有的实例变量共用同一个 static 变量.

对于静态类，只能用于嵌套类内部类中。

Reference

- Static class in Java - GeeksforGeeks

Static 相关问题

Static 关键字是什么意思？

Static 关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例的情况下直接被访问

是否可以 override 一个 static 的方法？

不能被覆盖. 因为方法覆盖是基于运行时动态绑定的, 而 static 方法是编译时静态绑定的.

一个 static 方法内部调用非 static 方法?

不可以. 因为非 static 方法是要与对象关联在一起的, 须创建一个对象的实例后, 才可以在该对象上进行方法调用, 而static方法调用时不需要创建对象, 可以直接调用. 也就是说, 当一个 static 方法被调用时, 可能还没有创建任何实例对象, 如果从一个 static 方法中发出对非 static 方法的调用, 那个非 static 方法是关联到哪个对象上的呢? 这个逻辑无法成立, 所以, 一个 static 方法内部发出对非 static 方法的调用.

是否可以在 static 环境中访问非 static 变量？

同上

Singleton 单例模式

Java中单例模式定义：“一个类有且仅有一个实例，并且自行实例化向整个系统提供。”

```
public class Singleton {
    private Singleton() {
        // do something
    }
    private static class SingletonHolder {
        private static final Singleton INSTANCE = new Singleton();
    }
    public static final Singleton getInstance() {
        return SingletonHolder.INSTANCE;
    }
}
```

多选题注意

- 一是单例模式的类只提供私有的构造函数
- 二是类定义中含有一个该类的静态私有对象
- 三是该类提供了一个静态的公有的函数用于创建或获取它本身的静态私有对象。

Reference

- 深入浅出单实例Singleton设计模式 | 酷壳 - CoolShell.cn

equals() 与 hashcode()

Equal

如果需要比较对象的值，就需要equal方法了. 看一下JDK中equal方法的实现：

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

也就是说，默认情况下比较的还是对象的地址. 所以如果把对象放入Set中等操作，就需要重写eqaul方法了

重写之后的 equals() 比较的就是对象的内容了

hashcode

When inserting an object into a hastable you use a key. The hash code of this key is calculated, and used to determine where to store the object internally. When you need to lookup an object in a hashtable you also use a key. The hash code of this key is calculated and used to determine where to search for the object.

The hash code only points to a certain "area" (or list, bucket etc) internally. Since different key objects could potentially have the same hash code, the hash code itself is no guarantee that the right key is found. The hashtable then iterates this area (all keys with the same hash code) and uses the key's equals() method to find the right key. Once the right key is found, the object stored for that key is returned.

So, as you can see, a combination of the hashCode() and equals() methods are used when storing and when looking up objects in a hashtable.

If equal, then same hash codes too. Same hash codes no guarantee of being equal.

== 和 equal 的区别

- **==** 比较引用的地址
- **equel** 比较引用的内容 (Object 类本身除外)

```
String obj1 = new String("xyz");
String obj2 = new String("xyz");

// If String obj2 = obj1, the output will be true

if(obj1 == obj2)
    System.out.printlln("obj1==obj2 is TRUE");
else
    System.out.println("obj1==obj2 is FALSE");

// It will print obj1==obj2 is False
// If String obj2 = obj1, the output will be true
```

默认的, equals() 方法实际上和 “==” 在 object 类里是一样的. 但是这个方法在每一个子类里都会被覆写用来比较引用的内容 (因为每个类都继承了 object 类并覆写了这个方法)

```
String obj1 = new String("xyz");
String obj2 = new String("xyz");

if(obj1.equals(obj2))
   System.out.printlln("obj1==obj2 is TRUE");
else
  System.out.println("obj1==obj2 is FALSE");

 Resultat: obj1==obj2 is TRUE
```

所有类的基类是哪个类?

java.lang.Object

Does Java support multiple inheritance?

Java doesn't support multiple inheritance.

Path 与 Classpath?

Path 和 Classpath 是操作系统的环境变量.

- Path 定义了系统可以在哪里找到可执行文件(.exe)
- classpath 定义了 .class 文件的位置.

反射机制

JAVA反射机制是在运行状态中, 对于任意一个类, 都能够知道这个类的所有属性和方法; 对于任意一个对象, 都能够调用它的任意一个方法和属性; 这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制.

主要作用有三：

1. 运行时取得类的方法和字段的相关信息。
2. 创建某个类的新实例(.newInstance())
3. 取得字段引用直接获取和设置对象字段，无论访问修饰符是什么。

用处如下：

1. 观察或操作应用程序的运行时行为。
2. 调试或测试程序，因为可以直接访问方法、构造函数和成员字段。
3. 通过名字调用不知道的方法并使用该信息来创建对象和调用方法。

final 关键字

final 类是不能被继承的 这个类就是最终的了 不需要再继承修改 比如很多 java 标准库就是 final 类

final 方法不能被子方法重写

final + static 变量表示常量

一个 .java 源文件是否可以包含多个类

可以得

但只能有一个是 public 的类 而且这个 public 类必须与文件名一样

& 与 &&

都可以表示逻辑与 and, 但是 && 具有短路功能 第一个表达式错了 第二个就被忽略了。`&` 的表达式是先计算后求与。

除此外 & 可以用作位运算符

`|` 也有类似差异。

java - Difference between & and && - Stack Overflow

int 与 integer

int 是数据类型

interger 是 int 的封装类

int 默认值为 0 interger 默认值为 null 所以 interger 可以用来判断变量是否赋值 即 null 和 0 的区别

integer 通过 == 比较

```
Integer a=10;
Integer b=10;
Integer c=new Integer(10);
Integer d=new Integer(10);

System.out.println(a==b);
System.out.println(c==d);

System.out.println(a.equals(b));
System.out.println(c.equals(d));

System.out.println(a.equals(c));
```

结果为

```
true
false
true
true
true
```

== 比较的是对象的引用 当且仅当比较的两个引用指向同一对象才返回true

再看一个例子

```
Integer a = 127;
Integer b = 127;
Integer c = 128;
Integer d = 128;
System.out.println(a == b);
System.out.println(c == d);
```

结果为

```
true
false
```

Integer i = XXX 看看Integer 的源代码就知道了, 其实就是Integer 把-128-127(一个字节的二进制补码) 之间的每个值都建立了一个对应的Integer 对象, 类似一个缓存. 由于Integer 是不可变类, 因此这些缓存的Integer 对象可以安全的重复使用. Integer i = XXX, 就是Integer i =

Interger.valueOf(XXX), 首先判断XXX 是否在-128-127 之间, 如果是直接return 已经存在的对象, 所以是同一个引用. 否则就只能 new 一个了, 那就是不同的引用了.

Reference

- java - Why does 128==128 return false but 127==127 return true in this code? - Stack Overflow

作用域的区别

作用域	当前类	同一个 **package**	子孙类	其他 **package**
public	O	O	O	O
protected	O	O	O	X
friendly	O	O	X	X
private	O	X	X	X

异常

异常是指java程序运行时（非编译）所发生的非正常情况或错误

Java使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该对象中包含有异常的信息

Java对异常进行了分类，所有异常的根类为java.lang.Throwable

Throwable下面又派生了两个子类：Error和Exception

error 和 exception?

Error表示应用程序本身无法克服和恢复的一种严重问题, 程序只有死的份了, 例如, 说内存溢出和线程死锁等系统问题

Exception 表示程序还能够克服和恢复的问题, 比如一个输入参数不对引起的异常. 其中又分为系统异常和普通异常

Checked 异常与 Runtime 异常

- Runtime exceptions 是 runtime 阶段碰到的异常. 在编译的时候不需要检查 (checked). 例如, 数组脚本越界（ArrayIndexOutOfBoundsException) , 空指针异常（NullPointerException）, 类转换异常（ClassCastException).
- Checked exception 是在编译阶段的异常，并且强制检查.

编译器强制 **checked** 异常必须**try..catch**处理或用**throws**声明继续抛给上层调用方法处理, 这就是为什么叫**checked**异常, 而 Runtime 异常可以处理也可以不处理, 所以, 编译器不强制用try..catch处理或用throws声明, 所以 Runtime 异常也称为unchecked异常

异常选择题

题目一

下面哪个不对

1. RuntimeException is the superclass of those exceptions that must be thrown during the normal operation of the JVM
2. A method is not required to declare in its throws clause any subclasses of RuntimeExeption that might be thrown during the execution of the method but not caught.
3. An RuntimeException is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.
4. NullPointerException is one kind of RuntimeException

答案是 3

RuntimeException an unchecked exception. It doesn't need to be explicitly declared or catched.

题目二

特殊情况就是里面加 return

举个例子去理解

```
public int getNumber() {

    int a = 0;

    try {
        String s = "t"; -----------------------（1）
        a = Integer.parseInt(s);-----------（2）
        return a;
    } catch (NumberFormatException e) {
        a = 1;-----------------------------------（3）
        return a;-----------------------------（4）
    } finally {
        a = 2;-----------------------------------（5）
    }
}
```

1、程序中标记的代码的执行顺序? 2、改程序的最后返回值(外部调用时)?

程序按顺序从上到下执行到(2), 字符"t"转换成整数失败, 产生异常并被捕获, 于是对a赋值成1, 并将此值作为此方法的返回值（可以这么认为, 该方法有一个存放返回值的空间, 此时将1放在此处). 由于存在finally块, 在返回前将该方法的内部变量a修改成2. 所以程序将按标记的顺序执行, 外部调用该方法时得到的结果是1

先执行try或catch里里面的代码，然后再执行finally，再执行try或catch里面的return.

题目三

写出你最常见到的5个runtime exception

ClassCastException IllegalArgumentException NullPointerException IndexOutOfBoundsException ArrayIndexOutOfBoundsException

题目四

如果想要一个对象作为一个异常对象被抛出，应该怎么做. (就是自己 DIY 一个异常啦)

继承 Exception 类. 或者继成功 Exception 类里面的子类, 这样可以更加具体的表明哪一类异常.

如果我的类已经继承了其他的类,那应该怎么做?

那就没办法咯. Java 不支持多继承, 目前版本的 JDK 没有相关的接口.

处理异常的方法

1. try catch.
2. throws.

这两种方法有什么区别

第一种方法是自己处理异常.

第二种异常是把异常抛给调用这个方法的模块去处理. 一般 Java 的库就是怎么处理的.

每一个 try 都必须有一个 catch 吗?

不是必须的. 至少要有一个 catch 或者 finally 块.

如果在 try 模块里最后加了个 return, finally 模块还会执行吗?

是的. finally 模块会先执行再 return.

如果换成 System.exit (0)?

那就不会了. System.exit (0) 时. 会立马跳出程序.

try catch finally 的执行顺序

特殊情况就是里面加 return

举个例子去理解

```
public int getNumber() {

    int a = 0;

    try {
        String s = "t"; ------------------------（1）
        a = Integer.parseInt(s);-----------（2）
        return a;
    } catch (NumberFormatException e) {
        a = 1;-----------------------------------（3）
        return a;-------------------------------（4）
    } finally {
        a = 2;-----------------------------------（5）
    }
}
```

1、程序中标记的代码的执行顺序？ 2、改程序的最后返回值（外部调用时）？

程序按顺序从上到下执行到（2），字符"t"转换成整数失败，产生异常并被捕获， 于是对a赋值成1，并将此值作为此方法的返回值（可以这么认为，该方法有一个存放返回值的空间，此时将1放在此处）。由于存在finally块，在返回前将该方法的内部变量a修改成2。 所以程序将按标记的顺序执行，外部调用该方法时得到的结果是1

先执行try或catch里里面的代码，然后再执行finally，再执行try或catch里面的return.

final, finally, finalize的区别

- final 用于声明属性,方法和类, 分别表示属性不可变, 方法不可覆盖, 类不可继承.
- finally 是异常处理语句结构的一部分，表示总是执行.
- finalize 是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等. JVM不保证此方法总被调用.

Programme

Article11

How could Java classes direct program messages to the system console, but error messages, say to a file?

The class System has a variable out that represents the standard output, and the variable err that represents the standard error device. By default, they both point at the system console. This how the standard output could be re-directed:

```
Stream st = new Stream(new FileOutputStream("output.txt"));

System.setErr(st);
System.setOut(st);
```

Gabage Collection

什么是**GC**

GC是垃圾收集的意思(Gabage Collection), 内存处理是编程人员容易出现问题的地方, 忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃, Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的, Java语言没有提供释放已分配内存的显示操作方法.

垃圾回收器的基本原理是什么**?**

当程序员创建对象时，GC就开始监控这个对象的 地址、大小以及使用情况. 通常，GC采用有向图的方式记录和管理堆（heap）中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的".当GC确定一些对象为"不可达"时(比如设置为 null)，GC就有责任回收这些内存空间.

有什么办法主动通知虚拟机进行垃圾回收**?**

可以.程序员可以手动执行System.gc(),通知GC运行,但是Java语言规范并不保证GC一定会执行. 这个选择题的时候有考.

heap 和 stack

java的内存分为两类：

- 堆内存 heap
- 栈内存 stack

stack 是指程序进入一个方法时, 会为这个方法单独分配一块私属存储空间, 用于存储这个方法内部的局部变量, 当这个方法结束时, 分配给这个方法的栈会释放, 这个栈中的变量也将随之释放.

heap 一般用于存放不放在当前方法栈中的那些数据，例如, 使用 new 创建的对象都放在堆里, 所以, 它不会随方法的结束而消失. 方法中的局部变量使用 final 修饰后, 放在堆中, 而不是栈中.

GC 就一定能保证内存不溢出吗?

Non. 程序员可能创建了一个对象, 以后一直不再使用这个对象, 这个对象却一直被引用, 这个对象无用但是却无法被垃圾回收器回收的

字节流与字符流

- 字节流继承于InputStream OutputStream
- 字符流继承于InputStreamReader OutputStreamWriter

字符流使用了缓冲区 (buffer),而字节流没有使用缓冲区

底层设备永远只接受字节数据

字符是字节通过不同的编码的包装

字符向字节转换时，要注意编码的问题

Collection

Collection 的子类是 List 和 Set

ArrayList 和 Vector

这两个类都实现了List接口(List接口继承了Collection接口).

他们都是有序集合,即存储在这两个集合中的元素的位置都是有顺序的,相当于一种动态的数组

并且其中的数据是允许重复的

ArrayList与Vector的区别

- Vector是线程安全的, 也就是线程同步的, 而ArrayList是线程序不安全的. 对于Vector&ArrayList, Hashtable&HashMap, 要记住线程安全的问题, 记住Vector与Hashtable是旧的, 是java一诞生就提供了的, 它们是线程安全的, ArrayList与HashMap是java2时才提供的, 它们是线程不安全的.
- ArrayList与Vector都有一个初始的容量大小, 当存储进它们里面的元素的个数超过了容量时, 就需要增加ArrayList与Vector的存储空间, Vector默认增长为原来两倍,而ArrayList的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的1.5倍）.ArrayList与Vector都可以设置初始的空间大小, Vector还可以设置增长的空间大小, 而ArrayList没有提供设置增长空间的方法.

总结：即**Vector**增长原来的一倍**,ArrayList**增加原来的**0.5**倍**. Vector** 线程安全**, ArrayList** 不是**.**

HashMap 和 Hashtable

Hashtable是基于陈旧的Dictionary类的 HashMap是Java 1.2引进的Map接口的一个实现

Hashtable是线程安全的，也就是说是同步的 而HashMap是线程序不安全的，不是同步的

只有HashMap可以让你将空值null作为一个表的条目的key或value. 但是 HashTable 不允许

HashMap HashTable LinkedHashMap TreeMap

不允许键重复，值可以重复。

HashMap是一个最常用的Map, 它根据键的hashCode值存储数据, 根据键可以直接获取它的值, 具有很快的访问速度. HashMap最多只允许一条记录的键为null, 不允许多条记录的值为null. HashMap不支持线程的同步, 如果需要同步, 可以用Collections.synchronizedMap(HashMap map)方法使HashMap具有同步的能力.

Hashtable与HashMap类似, 不同的是: 它不允许记录的键或者值为空; 它支持线程的同步.

LinkedHashMap保存了记录的插入顺序, 在用Iteraor遍历LinkedHashMap时, 先得到的记录肯定是先插入的. 在遍历的时候会比HashMap慢. 有HashMap的全部特性.

TreeMap能够把它保存的记录根据键排序, 默认是按升序排序, 也可以指定排序的比较器. 当用Iteraor遍历TreeMap时, 得到的记录是排过序的. TreeMap的键和值都不能为空.

Collection 相关问题

题目一

```
You need to store elements in a collection that guarantees that no duplicates are stor
ed and all elements can be access in nature order, which interface provies that capabi
liy?

A. java.util.Map
B. java.util.Collection
C. java.util.List
D. java.util.Set
```

答案 D

题目二

```
List, Set, Map是否继承自Collection接口,它们有什么区别?
```

List，Set是，Map不是

Set 不允许有重复的元素.且没有顺路 Set取元素时,没法说取第几个,只能以Iterator接口取得所有的元素,再逐一遍历各个元素.

List表示有先后顺序的集合并且允许重复

Map与List和Set不同，存储一对key/value，不能存储重复的key

题目三

```
public static void main(){
    Map<String,String> map = new HashMap<String,String>();
    map.out(String.valueOf(System.currentTimeMillis())+"a",1);
    map.out(String.valueOf(System.currentTimeMillis())+"a",2);
    map.out(String.valueOf(System.currentTimeMillis())+"a",3);
    for(Map.Entry<String,String> entry : map.entrySet()){
        System.out.printf(entry.getValue());
    }
}
```

输出顺序是 123顺序无法确定. Map 中的键是 Set. Set 顺序是随机的.

Multi-Thread

sleep() 和 wait() 的区别

举个例子

```
sleep(1000)
```

会把把线程放到一边, 直到整整一秒之后才再次启动

```
wait(1000)
```

则是把线程放到一边至多一秒. 如果碰到 notify() 或者 notifyAll() 就会提前启动.

而且 wait() 方法是在 Object 类里. 而 sleep() 是在 Thread 类里.

同步 synchronized

举个例子

```
public class Synchronized Counter {
    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized void decrement() {
        c--;
    }

    public synchronized int value() {
        return c;
    }
}
```

如果 count 是这个类的实例化将有两个效果:

- 不可能同时调用同一个对象的同一个方法, 防止造成冲突.同一时间只有一个线程可以调用这对象的同步方法.比如在一个账户里同时存钱和转账.
- 当一个同步方法退出时, *它会和随后一个同步方法的调用自动建立happens-before关系. 这保证了所有线程都知道对象的状态改变了.

如何实现 muliti-thread?

- 继续Thread类
- 实现Runable接口

Thread 与 Runnable?

实现Runnable接口比继承Thread类所具有的优势：

- 适合多个相同的程序代码的线程去处理同一个资源

- 可以避免java中的单继承的限制

- 增加程序的健壮性，代码可以被多个线程共享，代码和数据独

Transient 关键字

当持久化对象时, 可能有一个特殊的对象数据成员, 我们不想用 serialization 机制来保存它. 为了在一个特定对象的一个域上关闭serialization, 可以在这个域前加上关键字transient.

preemptive scheduling 和 time slicing?

preemptive scheduling, 优先级别最高的任务会被执行, 除非它进入等待状态或者死了或者一个更高优先权的任务进来.

time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

一个线程的初始状态是什么?

一个线程被创建和开始之后是 “Ready” 状态.

synchronized method 和 synchronized statement?

Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

守护线程 daemon thread?

守护线程, 是指在程序运行的时候在后台提供一种通用服务的线程, 比如垃圾回收线程就是一个很称职的守护者, 并且这种线程并不属于程序中不可或缺的部分. 因此, 当所有的非守护线程结束时, 程序也就终止了, 同时会杀死进程中的所有守护线程. 反过来说, 只要任何非守护线程还在运行, 程序就不会终止.

用户线程和守护线程两者几乎没有区别, 唯一的不同之处就在于虚拟机的离开: 如果用户线程已经全部退出运行了, 只剩下守护线程存在了, 虚拟机也就退出了.

将线程转换为守护线程可以通过调用Thread对象的setDaemon(true)方法来实现.

所有的线程都必须实现哪个方法?

run() 方法, 不管是继承 Thread 还是实现 Runnable 接口.

Visitor Pattern

Visitor -- switch case

Problem on chain

字符串基础问题

题目一

```
public class Test{
    public static void main(String[] args){
        String s1 = "abc";
        String s2 = s1;
        String s3 = new String("abc");
        String s4 = new String("abc");
        String s5 = "abc";

        System.out.println(s1==s5);
        System.out.println(s1==s2);
        System.out.println(s1.equals(s2));
        System.out.println(s3==s4);
        System.out.println(s1.equals(s4));
        System.out.println(s3.equals(s4));
    }
}
```

输出是 true true true false true true

记住 == 比较引用. equals 比较值. String 对象会创建一个字符串池 (a pool of string), 如果当前准备新创建的字符串对象的值在这个池子中已经存在，那么就不会生成新对象, 而是复用池中已有的字符串对象. 不过, 只有采用 Object s = “Hello” 方式 (而非用”new“关键字) 声明 String 对象的时候这个规则才会被应用.

题目二

```
String s = "a" + "b" + "c" + "d" + "e"
```

创建了几个对象? 1个 赋值号右边都是常量,编译时直接储存它们的字面值, 在编译时直接把结果取出来面成了 "abcde"

题目三

```
# 创建了几个String Object?二者之间有什么区别？

    String s = new String("xyz");
```

两个或一个, ”xyz”对应一个对象, 这个对象放在字符串常量缓冲区, 常量”xyz”不管出现多少遍, 都是缓冲区中的那一个. New String每写一遍, 就创建一个新的对象, 它一句那个常量”xyz”对象的内容来创建出一个新String对象. 如果以前就用过’xyz’, 这句代表就不会创建”xyz”自己了, 直接从缓冲区拿.

题目四

```
String s1 = new String("777");
String s2 = "aaa777";
String s3 = "aaa" + "777";
String s4 = "aaa" + s1;
```

s2 == S3 : true s2 == S4 : false s2 == S4.intern() : true

题目五

```
String str = "ABCDEFGH";
String str1 = str.substring(3,5);
System.oout.println(str1);
```

输出是 DE substring 是前包括后不包括

题目六

```
执行后，原始的 String 对象中的内容到底变了没有？

    String s = "Hello";
    s = s + " world!";
```

没有. 因为 String 被设计成不可变**(immutable)**类, 所以它的所有对象都是不可变对象.

在这段代码中, s原先指向一个String对象, 内容是 "Hello",然后我们对s进行了+操作, 这时，s不指向原来那个对象了, 而指向了另一个 String对象, 内容为"Hello world!", 原来那个对象还存在于内存之中, 只是s这个引用变量不再指向它了.

题目七

执行后，输出是什么? 共创建了几个字符串对象?

```
   String s = " Hello ";
   s += " World ";
   s.trim( );
   System.out.println(s);
```

再次强调 String 是不可变**(immutable)**类. 所以共创建了三个对象. 最后输出是 " Hello World ". 只有 s = s.trim() 之后才是 "Hello World".

StringBuffer 相关问题

题目一

```
StringBuffer 和 StringBuilder
```

- StringBuilder 比 StringBuffer 快
- 当需要保证线程安全的时候用 StringBuffer
- StringBuffer 是 synchronized, StringBuilder 不是.

String 类一般被认为是不可改变的. 如果需要对一个String做许多修改就需要使用StringBuffer或者StringBuilder.

在Oracle里的定义就是 "A mutable sequence of characters."

另外需要注意 String 类是 final 类不可以被继承. 有时候会在考察 final 关键字的时候考这个.

数组相关问题

题目一

```
Which of the following are valid array declaration for srings of 50 chars?

A. char c[][];
B. String []s;
C. String s[50];
D. Object s[50];
```

答案 B

```
int [][] iArray;
int []iArray[];
intt iArray[][];
```

都是可以得, 但数组不能直接指定列数或者行数. 正确的方式应该在创建数组对象是.

比如

```
int iArray[][] = new int[3][4]
```

题目二

序列化 serialization

Serialization is a mechanism by which you can save the state of an object by converting it to a byte stream.

JAVA中实现serialization主要靠两个类:

- ObjectOuputStream
- ObjectInputStream

他们是JAVA IO系统里的OutputStream和InputStream的子类

自定义序列化的作用如下：

1. Persist only meaningful data.
2. Manage serialization between different versions of your class.
3. Avoid exposing the serialization mechanism to client API.

Reference

- The Java HotSpot: Customizing Java Serialization [Part 2]

如何序列化一个对象到一个文件?

要被序列化的实例所对应的类必须实现 Serializable 接口. 然后你可以把实例传递给 ObjectOutputStream, 同时 ObjectOutputStream 也必须连接至 fileoutputstream. 这样就会把一个对象储存到一个文件里.

必须实现 Serializable 接口的哪个方法?

Serializable 接口是一个空接口. 所以我们不实现它的任何方法.

如何控制 serialization 的过程?

Yes it is possible to have control over serialization process. The class should implement Externalizable interface. This interface contains two methods namely readExternal and writeExternal. You should implement these methods and write the logic for customizing the serialization process.

什么情况下要使用序列化?

Whenever an object is to be sent over the network, objects need to be serialized. Moreover if the state of an object is to be saved, objects need to be serilazed.

Externalizable 接口?

Externalizable is an interface which contains two methods readExternal and writeExternal. These methods give you a control over the serialization mechanism. Thus if your class implements this interface, you can customize the serialization process by implementing these methods.

序列化时引用的处理?

When an object is serialized, all the included objects are also serialized alongwith the original object

序列化时要注意什么?

One should make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

序列化时 static 域的处理?

There are three exceptions in which serialization doesnot necessarily read and write to the stream. These are

1. Serialization ignores static fields, because they are not part of auy particular state state.
2. Base class fields are only hendled if the base class itself is serializable.
3. Transient fields.

Initialization and Cleanup

Initialization and Cleanup

5. 初始化和清理

5.7 构造器初始化

5.7.1 初始化顺序

类内部变量定义的先后顺序决定了其初始化的顺序，并且会在任何方法（包括构造器，与顺序无关）被调用之前也会得到初始化。对于静态对象与非静态对象：先初始化静态对象，然后是非静态对象。

5.7.2 静态数据的初始化

静态数据只占用一份存储区域，static 关键字不能用于局部变量，因为它只能作用于域。如果一个域是静态的基本类型域且未对其初始化，那么它就会获得基本类型的标准初值；如果是一个对象引用，则初始化为 null.

静态初始化只有在必要时才会进行，且只被初始化一次，即如果不创建相应的对象或是引用相应的静态对象，那么则不会被初始化。

对象创建过程：

1. 构造器实际上也是静态方法。Java 解释器首先查找类路径定位相应 class 文件。
2. 载入 class 文件，执行静态初始化，静态初始化只在类对象首次加载的适合进行一次。
3. 使用 new 创建对象时首先将在堆上为对象分配足够的存储空间。
4. 存储空间清零，故其所有基本类型数据置为默认值。
5. 执行所有定义处的初始化动作。
6. 执行构造器。

Java Data Types - Java 数据类型

JVM 可以操作的数据类型分为两类: primitive types 和 reference types. 类型检查通常在编译期完成，不同指令操作数的类型可以通过虚拟机的字节码指令本身确定。

Primitive type

JVM 所支持的基本数据类型有：数值类型(Numeric types), 布尔类型(Boolean type) 和 returnAddress 类型。其中数值类型又可以分为整型和浮点型两种。

- 整型：byte(8 bit), short(16 bit), int(32 bit), long(64 bit), char(16 bit unsigned)
- 浮点型：float(32 bit), double(64 bit)
- 布尔型：boolean 通常用 int 型表示，Oracle 中用 byte 表示
- returnAddress：一条字节码指令的操作码

Reference type

引用类型分为三种：Class Types, Array Types 和 Interface Types, 这些引用类型的值分别由类实例、数组实例和实现了某个接口的类实例或者数组实例动态创建。引用类型中有一特殊的值 `null`, 引用类型的默认值就是 null.

形式参数传递

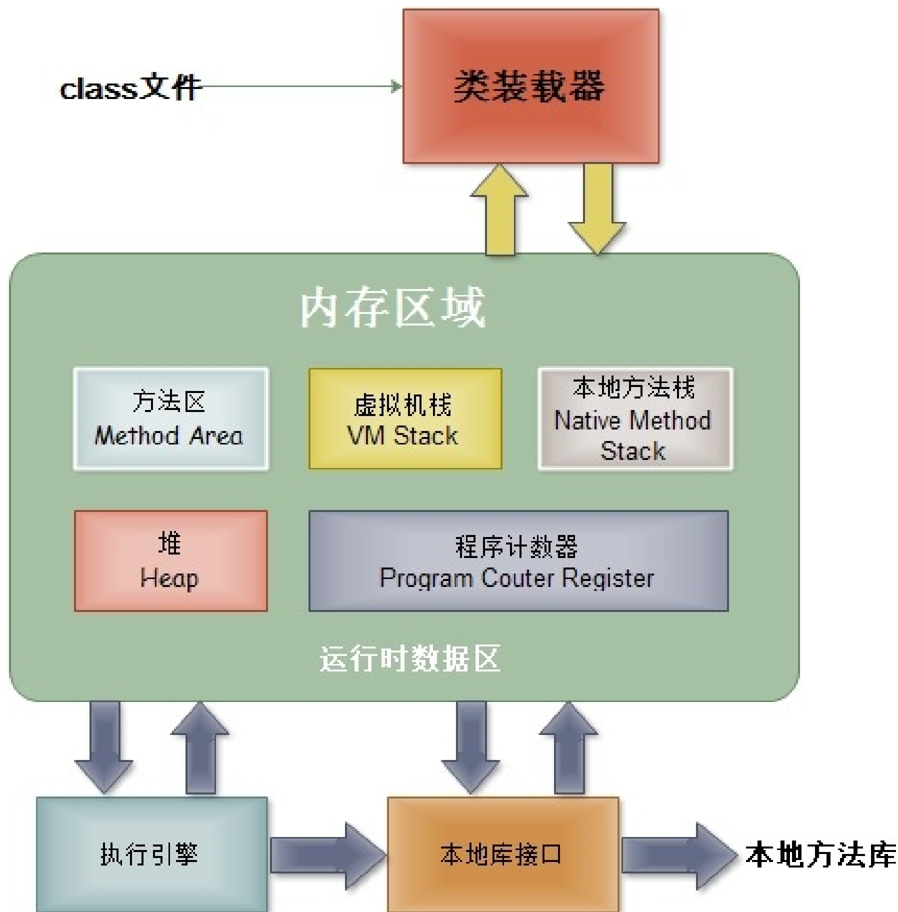
基本类型作为形式参数传递不会改变实际参数，引用类型作为形式参数传递会改变实际参数。JDK1.5之后含有基本类型的包装类型，即自动拆装箱的功能，故将基本类型的相应对象作为参数传递时会自动拆箱为基本类型，故也不改变实际参数的值。

Reference

- Chapter 2. The Structure of the Java Virtual Machine

Run-Time Data Areas - 运行时数据区域

JVM 运行时会有几个运行时数据区域，如下图所示。



The pc Register - 程序计数器

线程私有内存，保存当前线程所执行的字节码的行号指示器，这里和计算机组成原理中的计数器不太一样，计组中的 PC 指的是下一条要执行的指令的地址。JVM 中常有多个线程执行，故每条线程都需要有一个独立的程序计数器。

如果线程执行的是 Java 方法，哪儿计数器记录的就是正在执行的虚拟机字节码指令的地址；如果执行的是 Native 方法，这个计数器则为空。

P.S. 这块内存无 `OutOfMemoryError`

Java Virtual Machine Stacks - Java 虚拟机栈

线程私有，虚拟机栈描述的是 Java 方法执行的内存模型，每个方法在执行时会创建一个栈帧，栈帧中保存有局部变量表、操作数栈、动态链接和方法出口等。粗略来讲 Java 内存区分为堆和栈，实际上『栈』指的往往是虚拟机栈中的局部变量表部分。

局部变量表中存放了编译期可知的各种基本数据类型、对象引用类型和 returnAddress 类型。方法运行期间局部变量表大小不变。

Native Method Stacks - 本地方法栈

和虚拟机栈类似，不过区别在于虚拟机栈为 Java 方法（字节码）服务，而本地方法栈为 Native 方法服务（类似 C 语言中的栈）。具体实现可将这两者合二为一。

Heap - 堆

堆是被所有线程共享的一块内存区域。一般来说所有的对象实例和数组都要在堆上分配，但一些优化技术导致不一定所有对象实例都在堆上分配。

Method Area - 方法区

各线程共享的一块内存区域，和操作系统中进程中的『文本段』有些类似，用于存储虚拟机加载的类信息、常量、静态常量和即时编译器编译后的代码数据等。

Run-Time Constant Pool - 运行时常量池

这一部分是方法区的一部分，用于保存 Class 文件中编译期生成的字面值和符号引用。

直接内存

这一部分并不是虚拟机运行时的数据区域，用于 Native 函数分配堆外内存，提高性能用（不必在操作系统堆和 Java 堆复制数据）。

Reference

- 《深入理解 Java 虚拟机》
- Java 内存区域详解 - SegmentFault
- Chapter 2. The Structure of the Java Virtual Machine

What are considered as a web component?

什么是 J2EE?

An environment for developing and deploying enterprise applications.

The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitier, web-based applications.

J2EE 应用的四个部分?

我更喜欢说是层

- 客户端层
- web层 (Servlet and JSP)
- 业务层 (JavaBeans)
- 企业信息系统层（Enterprise Information System tier) 或者叫Resource adapter 包含数据库等

What does application client module contain?

The application client module contains:

- class files
- an application client deployment descriptor.

Application client modules are packaged as JAR files with a .jar extension.

What does web module contain?

The web module contains:

- JSP files
- class files for servlets
- GIF and HTML files
- a Web deployment descriptor.

Web modules are packaged as JAR files with a **.war** (Web ARchive) extension.

J2EE客户端有哪些类型？

- Applets
- Application clients
- Java Web Start-enabled clients, by Java Web Start technology.
- Wireless clients, based on MIDP technology.

Hibernate是什么?

- object-relational *
- In hibernate we can write HQL instead of SQL which save developers to spend more time on writing the native SQL.
- 我们能像处理 Java 对象一样处理数据库
- 所以可以在处理的时候加入 Java 语言的特性 比如继承啊 多态啊 。。
- Hibernate also allows you to express queries using java-based criteria .

什么是事务 - transaction

事务是应用程序中一系列严密的操作，所有操作必须成功完成，否则在每个操作中所做的所有更改都会被撤消。例如，将资金从支票帐户转到储蓄帐户中是一项事务，按步骤如下进行：

检查支票帐户是否有足够的资金来支付此转帐操作。

如果支票帐户中有足够的资金，则将该笔资金记入此帐户的借方。

将这些资金记入储蓄帐户的贷方。

将此次转帐记录到支票帐户日志中。

将此次转帐记录到储蓄帐户日志中。

什么是 servlet?

Servlets 是服务器端的部件

是纯的 java 对象

设计用于多种协议 特别是HTTP

创建 servlet

Servlet 在容器中运行时，其实例的创建及销毁等是由容器进行控制.

Servlet 的创建有两种方法。

1. 客户端请求对应的 Servlet 时，创建 Servlet 实例.大部分Servlet 都是这种 Servlet.
2. 通过在 web.xml 中设置load-on-startup来创建servlet实例，这种实例在Web 应用启动时，立即创建 Servlet 实例

Servlet 必须实现什么接口?

Servlet Interface

servlet 生命周期?

1. 读取 Servlet 类
2. 创建 Servlet 实例
3. Web 容器调用 Servlet 的 init() 方法
4. 响应客户端请求通过Servlet中service()方法中相应的doXXX()方法
5. 调用 Servlet 的 destroy()

JSP

JavaServer Pages (JSP)

delivering **dynamic content** to web applications in a portable, secure and well-defined way.

The JSP Technology allows us to use HTML, Java, JavaScript and XML in a single file to create high quality and fully functionaly User Interface components for Web Applications.

JSP 的生命周期?

1. Compilation
2. Initialization
3. Execution
4. Cleanup

详细解释

Compilation: When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

编译的三个步骤:

1. Parsing the JSP.
2. Turning the JSP into a servlet.
3. **Compiling the servlet.**

Initialization: When a container loads a JSP it invokes the jspInit() method before servicing any requests

Execution: Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

Cleanup: The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.The jspDestroy() method is the JSP equivalent of the destroy method for servlets.

JSP language

- <%@ directive %>
- <%! declaration %>
- <%= expression %>
- <% code fragment %>
- <%-- comment --%>

What are JSP declarations?

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

```
<%! declaration; [ declaration; ]+ ... %>
```

JSP expressions?

A JSP expression element contains a scripting language expression that is evaluated, converted to a **String**, and inserted where the expression appears in the JSP file.

```
<%= expression %>
```

JSP Directives?

A JSP directive affects the overall structure of the servlet class. 通俗的讲就是告诉引擎如何处理其余JSP页面

Types of directive tags?

<%@ page ... %> : Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

<%@ include ... %> : Includes a file during the translation phase.

<%@ taglib ... %> : Declares a tag library, containing custom actions, used in the page.

两者的区别

Both the tags include information from one JSP page in another. The differences are:

< jsp : include page = ... >

This is like a function call from one jsp to another jsp. It is executed each time the client page is accessed by the client. This approach is useful while modularizing a web application. If the included file changes then the new content will be included in the output automatically.

<% @ include file = ... >

In this case the content of the included file is textually embedded in the page that have < % @ include file=".."> directive. In this case when the included file changes, the changed content will not get included automatically in the output. This approach is used when the code from one jsp file required to include in multiple jsp files.

注释的类型

被隐藏的注释 (只有开发人员能够看到):

```
< % - - This is a hidden comment - - % >
```

被输出的注释 (用户可以直接通过查看网页源代码看到) :

```
< ! - - This is an output comment - - >
```

What do you understand by JSP Actions?

JSP actions are XML tags that direct the server to use existing components or control the behavior of the JSP engine.

There are six JSP Actions:

- < jsp : include / >
- < jsp : forward / >
- < jsp : plugin / >
- < jsp : usebean / >
- < jsp : setProperty / >
- < jsp : getProperty / >

和 response.sendRedirect(url)?

jsp:forward 把 request object 传递给服务器里的另外一个 servlet 或者 JSP. 新的 servlet 或者 JSP 继续处理同一个 request and the 但浏览器并不知道这些. 在浏览器里的 URL 不变.

response.sendRedirect() 创建了一个新的 request object, 不携带旧的 request object 的信息. The first request handler JSP page tells the browser to make a new request to the target servlet or JSP page. 在浏览器里的 URL 改变.

Scope for the < jsp : useBean > tag?

< jsp : useBean > tag is used to use any java object in the jsp page.

a) page b) request c) session d) application

JSP translation?

Conversion of the JSP Page into a Java Servlet. This class is essentially a servlet class wrapped with features for JSP functionality.

Ear, Jar 和 War 文件的区别?

- Jar files are intended to hold generic libraries of Java classes, resources, etc.
- War files are intended to contain complete Web applications.
- Ear files are intended to contain complete enterprise applications.

URI 和 URL?

URIs identify and URLs locate (on network); however, locators are also identifiers.

So all URLs are URIs (actually not quite - see below), and all URNs are URIs - but URNs and URLs are different, so you can't say that all URIs are URLs.

DAO

Spring

什么是 Spring?

Spring 是 Java EE 的是一个轻量级的开源框架.

使 J2EE 开发更容易

通过实现基于POJO的编程模型

Spring 的核心 design pattern 是 IOC

what are benefits of using spring?

Lightweight: Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.

Inversion of control (IOC): Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.

Aspect oriented (AOP): Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

Container: Spring contains and manages the life cycle and configuration of application objects.

MVC Framework

Transaction Management: Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Exception Handling: Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

Spring 都有哪些模块?

The Core container module O/R mapping module (Object/Relational) DAO module Application context module Aspect Oriented Programming Web module MVC module

什么是 Spring 的配置文件?

Spring 的配置文件是一个 XML 文件.

这个文件包含类的或者说 bean 的信息以及它们是如何配置的

什么是依赖注入 - Dependency Injection?

反转控制 Inversion of Control (IoC) 或者叫依赖注入 Dependency Injection

is a general concept, and it can be expressed in many different ways and Dependency Injection is merely one concrete example of Inversion of Control.

IoC 的类型?

- Constructor-based dependency injection:
- Setter-based dependency injection:

你更倾向于哪种 DI

采用以设置注入为主，构造注入为辅. 对于依赖关系无须变化的注入, 尽量采用构造注入;而其他的依赖关系的注入, 则采用设置注入.

对于依赖关系无须变化的Bean，构造注入更有用处;因为没有setter方法，所有的依赖关系全部在构造器内设定，因此，不用担心后续代码对依赖关系的破坏.安全性高.

setter : 创建完对象之后再同过set（）方法进行设定. 对于复杂的依赖关系, 如果采用构造注入, 会导致构造器过于臃肿, 难以阅读.

IoC 有什么好处?

- 减少代码
- 是应用更容易测试
- 松耦合 (Loose coupling) 和最小的侵入性
- IOC containers support eager instantiation and lazy loading of services.

IoC container 是什么?

管理 bean 的生命周期 (从创建,配置等等再到摧毁)

通过 dependency injection (DI) 管理构成一个应用各个部件

IoC containers 类型?

- Bean Factory container
- Spring ApplicationContext Container

ApplicationContext 的实现都有哪些?

- FileSystemXmlApplicationContext
- ClassPathXmlApplicationContext
- WebXmlApplicationContext

Bean Factory 与 ApplicationContext 的区别?

Bean Factory 只提供基础的 DI 支持

Application contexts 提供了处理 text messages 的功能 读取 file resources 的功能, 比如图片.

发布事件给已经注册为 listener 的 bean.

什么是 Spring beans?

一个 bean 是被实例化, 组装, 以及由Spring IoC容器管理的对象.

都有哪些 bean scope?

- singleton: Return a single bean instance per Spring IoC container
- prototype: Return a new bean instance each time when requested
- request: Return a single bean instance per HTTP request
- session: Return a single bean instance per HTTP session
- global-session: Return a single bean instance per global HTTP session

默认的是 **singleton**

Singleton bean 是线程安全的吗?

不是

PS 什么叫线程安全

一段代码，同时几个线程同时使用，结果都是正确的，就叫线程安全.

比如我们打开百度知道的首页,全世界很多人都在打开,都是正确的,证明百度知道首页的那段代码是线程安全的.

Bean lifecycle

1. Instantiate - 容器在 XML 文件里找到定义并实例化它们
2. Populate properties - 使用 DI 填充属性
3. Set Bean Name - If the bean implements BeanNameAware interface, spring passes the bean's id to setBeanName() method.
4. Set Bean factory - If Bean implements BeanFactoryAware interface, spring passes the beanfactory to setBeanFactory() method.
5. Pre Initialization - 也叫 postprocess. Spring 调用 postProcesserBeforeInitialization() 方法.
6. Initialize beans - If the bean implements IntializingBean,its afterPropertySet() method is called. If the bean has init method declaration, the specified initialization method is called.
7. Post Initialization - 调用 postProcessAfterInitialization() 方法
8. Ready to use - 现在可以用它们了.
9. Destroy - If the bean implements DisposableBean , it will call the destroy() method .

Annotation-based container configuration?

不用 XML 去描述 bean 的装配, 而是在类的代码里使用注释来配置

How can you inject Java Collection in Spring?

: injecting a list of values, 允许重复.

: This helps in wiring a set of values 不允许重复.

: name-value pairs where name and value can be of any type.

:name-value pairs where the name and value are both Strings.

什么是自动装配

Spring 自动解决 bean 之间的关系. 通过检查 BeanFactory 中的内容， 而无需使用 `<constructor-arg>` 和 `<property>` 元素

什么是 AOP?

面向方面的编程 (Aspect-oriented programming),

它可以运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术. 比如检测某个模块的运行时间. 加入额外的功能 (introduce)

下面是专业的所法:

利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率

主要的功能是：日志记录，性能统计，安全控制，事务处理，异常处理等等。

通知的类型?

before: Run advice before the a method execution.

after: Run advice after the a method execution regardless of its outcome.

after-returning: Run advice after the a method execution only if method completes successfully.

after-throwing: Run advice after the a method execution only if method exits by throwing an exception.

around: Run advice before and after the advised method is invoked.

Join point?

是我们刻意加入 AOP 切面的位置. 实际上它是程序里某个动作发生的地方, 比如某个程序的执行.

Pointcut

This is a set of one or more joinpoints where an advice should be executed.

Introduction?

An introduction allows you to add new methods or attributes to existing classes.

How do you provide configuration metadata to the Spring Container?

How do add a bean in spring application?

Can you inject null and empty string values in Spring?

@Autowired @Inject @Resource

@Autowired and @Inject

```
Matches by Type
Restricts by Qualifiers
Matches by Name

AutowiredAnnotationBeanPostProcessor
```

@Resource

```
Matches by Name
Matches by Type
Restricts by Qualifiers

CommonAnnotationBeanPostProcessor
```

Hibernate

get and load

get vs load is one of the most frequently asked Hibernate Interview question, since correct understanding of both get() and load() is require to effectively using Hibernate. Main difference between get and load is that, get will hit the database if object is not found in the cache and returned completely initialized object, which may involve several database call while load() method can return proxy, if object is not found in cache and only hit database if any method other than getId() is called. This can save lot of performance in some cases. You can also see difference between get and load in Hibernate for more differences and detailed discussion on this question.

Read more: http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html#ixzz3BJkBQ5vg

What is SessionFactory in Hibernate?

SessionFactory 是一个创建 hibernate Session 对象的工厂.

它可以作为单一的 data store 及也是线程安全的，使多个线程可以使用相同的 SessionFactory.

一个 Java JEE 应用只有一个 SessionFactory 如果只有一个数据库的话

当创建之后关于 Object/Relational mapping 的元数据是不能改的.

SessionFactory 是线程安全的吗?

是的

What is Session in Hibernate?

- Session 代表一个小单位的工作
- 它保持与数据库的连接
- 并且它们是非线程安全的

sorted 和 ordered collection

sorted collection是在内存中通过 java 比较器进行排序的 ordered collection是在数据库中通过 order by 进行排序的

建议使用 ordered collection 避免 OutOfMemoryError 问题.

What is the file extension used for hibernate mapping file?

filename.hbm.xml

hibernate 的三种状态

临时状态**(Transient):**

当new一个实体对象后, 这个对象处于临时状态, 即这个对象只是一个保存临时数据的内存区域, 如果没有变量引用这个对象, 则会被jre垃圾回收机制回收. 这个对象所保存的数据与数据库没有任何关系, 除非通过Session的save或者SaveOrUpdate把临时对象与数据库关联, 并把数据插入或者更新到数据库, 这个对象才转换为持久对象.

持久状态**(Persistent):**

持久化对象的实例在数据库中有对应的记录, 并拥有一个持久化表示（ID）. 对持久化对象进行delete操作后, 数据库中对应的记录将被删除, 那么持久化对象与数据库记录不再存在对应关系, 持久化对象变成临时状态. 持久化对象被修改变更后, 不会马上同步到数据库, 直到数据库事务提交. 在同步之前, 持久化对象是脏的（Dirty）.

游离状态**(Detached)**：

当Session进行了Close、Clear或者evict后, 持久化对象虽然拥有持久化标识符和与数据库对应记录一致的值, 但是因为会话已经消失, 对象不在持久化管理之内, 所以处于游离. 游离状态的对象与临时状态对象是十分相似的, 只是它还含有持久化标识.

Linux

查找文件

通过名字查找文件

```
find -name "nameFile"
```

但是分大小写.

如果不区分大小写用下面的命令:

```
find -iname "nameFile"
```

列出文件列表

```
ls
```

列出当前目录的所有内容

SQL

数据库管理系统在写入/更新资料过程中为保证事务是正确可靠的，其必须具备 ACID 特性。

ACID

- A: atomicity(原子性) - 一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。
- C: consistency(一致性) - 在事务开始之前和事务结束以后，数据库的完整性没有被破坏。即从一个一致性状态转换到另一个一致性状态。
- I: isolation(隔离性) - 当两个或者多个事务并发访问（此处访问指查询和修改的操作）数据库的同一数据时所表现出的相互关系。通常来说，一个事务所做的修改在最终提交以前，对其他事务是不可见的。
- D: durability(持久性) - 在事务完成以后，该事务对数据库所作的更改便持久地保存在数据库之中，并且是完全的。

符合 ACID 的典型例子有银行转账，整个过程称为一个事务，具有 ACID 特性。

Reference

- ACID - 维基百科，自由的百科全书
- 《高性能 MySQL》

设计一对一

一对一可以两个实体设计在一个数据库中.例如设计一个夫妻表，里面放丈夫和妻子

One to multi

一对多可以建两张表，将一这一方的主键作为多那一方的外键，例如一个学生表可以加一个字段指向班级(班级与学生一对多的关系)

multi to multi

多对多可以多加一张中间表，将另外两个表的主键放到这个表中（如教师和学生就是多对多的关系）

都使用过哪些join?

初学者说这2个 inner join (left/right) outer join

再牛逼点补充下下面2个 cross join self-join

inner join

INNER JOIN 关键字在表中存在至少一个匹配时返回行.如果没有匹配，就不会列出这些行.

例子

http://www.w3school.com.cn/sql/sql_join_inner.asp

Left/Right join

LEFT JOIN 关键字会从左表 (table_name1) 那里返回所有的行,即使在右表 (table_name2) 中没有匹配的行.

例子

RIGHT JOIN 关键字会右表 (table_name2) 那里返回所有的行,即使在左表 (table_name1) 中没有匹配的行.

例子

Full join

FULL JOIN 称为 FULL OUTER JOIN

FULL JOIN 关键字会从左表和右表那里返回所有的行, 即使没有匹配

合并

How can you combine two tables/views together? For instance one table contains 100 rows and the other one contains 200 rows, have exactly the same fields and you want to show a query with all data (300 rows).

```
SELECT column_name FROM table_name1
UNION
SELECT column_name FROM table_name2
```

注意

- UNION 内部的 SELECT 语句必须拥有相同数量的列.
- 列也必须拥有相似的数据类型.
- 每条 SELECT 语句中的列的顺序必须相同.

Union 和 Union all?

UNION 操作符选取不同的值. 如果允许重复的值，使用 UNION ALL.

Where 和 Having

WHERE 从句一般是在行的层级去筛选数据 (before grouping). HAVING 从句一般在 GROUP BY 之后所以是在 "groups" 的基础上删选.

更准确的说在 SQL 中增加 HAVING 子句原因是 WHERE 关键字无法与合计函数一起使用

因为在查询过程中聚合语句**(sum,min,max,avg,count)**要比**having**子句优先执行**.**而**where**子句在查询过程中执行优先级别优先于聚合语句**(sum,min,max,avg,count)**

What type of wildcards have you used?

首先说下什么是 wildcards

Wildcards are special characters that allow matching string without having exact match.

SQL 通配符必须与 LIKE 运算符一起使用

% 替代一个或多个字符

_ 仅替代一个字符

[charlist] 字符列中的任何单一字符

charlist 或者 [!charlist] 不在字符列中的任何单一字符

Scrum

- 迭代式开发 Le développement itératif
- 以人为核心

为什么说是以人为核心？

我们大部分人都学过瀑布开发模型，它是以文档为驱动的，为什么呢？因为在瀑布的整个开发过程中，要写大量的文档，把需求文档写出来后，开发人员都是根据文档进行开发的，一切以文档为依据；而敏捷开发它只写有必要的文档，或尽量少写文档，敏捷开发注重的是人与人之间，面对面的交流，所以它强调以人为核心。

什么是迭代？

迭代是指把一个复杂且开发周期很长的开发任务，分解为很多小周期可完成的任务，这样的一个周期就是一次迭代的过程；同时每一次迭代都可以生产或开发出一个可以交付的软件产品。

Scrum 中的三大角色

- 产品负责人（Product Owner）

主要负责确定产品的功能和达到要求的标准，指定软件的发布日期和交付的内容，同时有权力接受或拒绝开发团队的工作成果。

- 流程管理员（Scrum Master）

主要负责整个Scrum流程在项目中的顺利实施和进行，以及清除挡在客户和开发工作之间的沟通障碍，使得客户可以直接驱动开发。

- 开发团队（Scrum Team）

主要负责软件产品在Scrum规定流程下进行开发工作，人数控制在5~10人左右，每个成员可能负责不同的技术方面，但要求每成员必须要有很强的自我管理能力，同时具有一定的表达能力；成员可以采用任何工作方式，只要能达到Sprint的目标。

What's sprint?

Sprint是短距离赛跑的意思，这里面指的是一次迭代，而一次迭代的周期是1个月时间（即4个星期），也就是我们要把一次迭代的开发内容以最快的速度完成它，这个过程我们称它为Sprint。

How to scrum

1、我们首先需要确定一个Product Backlog, 这个是由Product Owner 负责的

2、Scrum Team根据Product Backlog列表, 做工作量的预估和安排

3、有了Product Backlog列表, 我们需要通过 Sprint Planning Meeting 来从中挑选出一个Story作为本次迭代完成的目标, 然后把这个Story进行细化, 形成一个Sprint Backlog；

4、每个成员根据Sprint Backlog再细化成更小的任务（细到每个任务的工作量在2天内能完成）

5、要进行 Daily Scrum Meeting（每日站立会议），每次会议控制在15分钟左右，每个人都必须发言，并且要向所有成员当面汇报你昨天完成了什么，并且向所有成员承诺你今天要完成什么，同时遇到不能解决的问题也可以提出，每个人回答完成后，要走到黑板前更新自己的 Sprint burn down（Sprint燃尽图）；

6、做到每日集成，也就是每天都要有一个可以成功编译、并且可以演示的版本；很多人可能还没有用过自动化的每日集成，其实TFS就有这个功能，它可以支持每次有成员进行签入操作的时候，在服务器上自动获取最新版本，然后在服务器中编译，如果通过则马上再执行单元测试代码，如果也全部通过，则将该版本发布，这时一次正式的签入操作才保存到TFS中，中间有任何失败，都会用邮件通知项目管理人员；

7、当一个Story完成，也就是Sprint Backlog被完成，也就表示一次Sprint完成，这时，我们要进行 Srpint Review Meeting（演示会议），也称为评审会议，产品负责人和客户都要参加（最好本公司老板也参加），每一个Scrum Team的成员都要向他们演示自己完成的软件产品

8、最后就是 Sprint Retrospective Meeting（回顾会议），也称为总结会议，以轮流发言方式进行，每个人都要发言，总结并讨论改进的地方，放入下一轮Sprint的产品需求中；

Continuous integration

un ensemble de pratiques utilisées en génie logiciel consistent à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Pour appliquer cette technique, il faut d'abord que :

le code source soit partagé les développeurs commit quotidiennement des tests d'intégration soient développés pour valider l'application il faut un outil d'intégration continue

好处: le test immédiat des unités modifiées la prévention rapide en cas de code incompatible ou manquant les problèmes d'intégration sont détectés et réparés de façon continue, évitant les problèmes de dernière minute une version est toujours disponible pour un test, une démonstration ou une distribution

statement 和 prepared statement?

Statement每次执行sql语句,数据库都要执行sql语句的编译. 最好用于仅执行一次查询并返回结果的情形，效率高于PreparedStatement.

Prepared statements offer better performance, as they are **pre-compiled**. Use when you plan to use the SQL statements **many times**.

Prepared statements are more secure because they use bind variables, which can prevent SQL injection attack.

Callable statement?

PreparedStatement继承自Statement CallableStatement继承自PreparedStatement

It's used when you want to access database stored procedures

Stored Procedure?

A stored procedure is a group of SQL statements that form a logical unit and perform a particular task. (粗俗的可以理解为一个定义好的方法, 提供输入就会得到对应的输出)

```
CallableStatement cs = con.prepareCall("{call MY_SAMPLE_STORED_PROC}");
ResultSet rs = cs.executeQuery();
```

What does the Class.forName("MyClass") do?

Loads the class MyClass.

Execute any static block code of MyClass.

Returns an instance of MyClass.

Connection Pooling ?

Connection Pooling is a technique used for reuse of physical connections and reduced overhead for your application.

Connection pooling functionality minimizes expensive operations in the creation and closing of sessions.

Database vendor's help multiple clients to share a cached set of connection objects that provides access to a database. Clients need not create a new connection every time to interact with the database.

What are the steps in the JDBC connection?

Step 1 : Register the database driver by using : Class.forName(\" driver classs for that specific database\");

Step 2 : Now create a database connection using :

Connection con = DriverManager.getConnection(url,username,password);

Step 3: Now Create a query using :

Statement stmt = Connection.Statement(\"select * from TABLE NAME\");

Step 4 : Exceute the query :

stmt.exceuteUpdate();